1. Data Modeling and Schema Design (5 points)

RDBMS schema

Entity 1: Departments

- Attributes:
  - department_id: Integer (Primary Key)
  - department_name: String (Unique, Not Null)

Entity 2: Instructors

- Attributes:
  - instructor_id: Integer (Primary Key)
  - name: String (Not Null)
  - email: String (Unique, Not Null)
  - department_id: Integer (Foreign Key → Departments)

Entity 3: Students

- Attributes:
  - student_id: Integer (Primary Key)
  - name: String (Not Null)
  - email: String (Unique, Not Null)
  - department_id: Integer (Foreign Key → Departments, Nullable)

Entity 4: Courses

- Attributes:
  - course_id: Integer (Primary Key)
  - course_name: String (Not Null)
  - course_code: String (Unique, Not Null)
  - department_id: Integer (Foreign Key → Departments)
  - Is_elective : (“Core” or “Elective”)

Entity 5: Enrollments

- Attributes:
  - enrollment_id: Integer (Primary Key)
  - student_id: Integer (Foreign Key → Students)
  - course_id: Integer (Foreign Key → Courses)

Entity 6: Course_Instructors

- Attributes:
  - enrollment_id: Integer (Primary Key)
  - student_id: Integer (Foreign Key → Students)
  - course_id: Integer (Foreign Key → Courses)

In this Schema, the tables are Normalized properly to Efficiently do WRITE operations. But the problem comes when there are many users who are requesting READ operations, in that case, we need to do computationally extensive JOIN operations to complete the request. To avoid this and to make the DB scalable and capable of handling large requests even with Large data, we will migrate the data to a NoSQL database(MongoDB).

I made a dummy dataset in Postgresql with the following features:

1. Total Departments - 8
2. Total Instructors - 100
3. Total Courses - 50
4. Total Students - 2000
5. Total Enrollments = each student have taken 25-30 courses so total enrollments are from 50,000 to 75,000
6. Total Courses_Instructos = each course is taught by 5 to 10 instructors, range(250 , 500)

MongoDb Schema

The Core Objective of making a document-based database is to improve scalability. In the Mongo DB schema, we have two aspects:

1. Embedding -> Embedding objects in the same document to query and retrieve the results faster instead.
2. Referencing -> Stores Object references of the ENTITIES stored in another document.

Embedding is preferred, but to an extent where it does not create redundancy or repetitiveness in the schema, if t does we use Referencing.
In MongoDB there is a limit of storing 16MB in a single Document. Thus we will use a Hybrid Approach to design the schema thus balancing the tradeoffs between these 2 concepts.

Document 1: Students

```
{
  "_id": "ObjectId",
  "student_id": "String",
  "name": "String",
  "email": "String",
 "Department": {     // An Embeddeding of departments to avoid JOIN to department table
          “department_id” : “string”
          “Department_name” : “string”
  },
  "enrollments": [{              // An Embedded list containing the Courses ids and name
      "course_id": "String",
      "course_name" : "String"
  }]
}
```

Document 2: Courses

```
{
  "_id": "ObjectId",
  "course_id": "String",
  "name": "String",
 “Category” : “String”,  // CORE or ELECTIVE
  "Department": {              // An Embeddeding of departments
          “department_id” : “string”
          “Department_name” : “string”
   },
  "instructor_id": "String",
  "enrollments": [{               // An Embedded list containing the references to the Students
          "student_id": "String"
          “Student_name” : “String”
   }]
  "instructors": [{              // An Embedded list containing the references to the instructors
      "instructor_id": "String",
      “instructor_name” : “String”
  }]
```

Document 3: Instructors

```
{
  "_id": "ObjectId",
  "instructor_id": "String",
  "name": "String",
 "Department": {           // An Embeddeding of departments
          “department_id” : “string”
          “Department_name” : “string”
   },
  "courses_taught": [{         // An Embedded list containing the Courses ids and name
          "course_id": "String"
          “Course_name” : “String”
  }]
}
```

According to the queries, I embedded both the id and name of any collection wherever required to give readable results.

Document 4: Departments

```
{
  "_id": "ObjectId",
  "department_id": "String",
  "name": "String",
  "courses": [{                          // Embedded the Course LIST with ids and names
        “Course_id” : “String”
        “Course_name” : “String”
   }],
  "instructors": [{                            // Embedded the object LIST with ids and names
        “Instructor_id” : “string”
        “Instructor_name” : “string”
   }]
  "students": [{                         // Embedded the students LIST with ids and names
        “Student_id” : “string”
        “Student_name” : “string”
   }]
}
```

Following DeNormalizations were made:

1. Embedding of Enrollments: The Enrollments entity from the relational schema was denormalized by embedding it into both the Students and Courses documents. This bidirectional embedding allows for efficient querying of both a student's courses and a course's students without additional database operations.
2. Instructor Courses List: An array of courses taught is added to the Instructor's document. This denormalization allows quick access to an instructor's courses without querying the Courses Collection.
   An list of Instructors is also added to Course collection, I have considered many to Many relationship between them, so I have to do bidirectional here too.
3. Department References List: Arrays of references to courses, instructors, and students are added to the Departments document. This structure allows for easy retrieval of all entities associated with a department without additional queries

There were 6 SQL tables initially , which was reduced to 4 documents(reports), this was done keeping in mind the queries given. Since there are no particular Complex queries, we didnt need to use referencing and thus all the queries will get executed without need of joining any 2 documents.

TradeOffs:

1. Potential data consistency issues- Suppose the department name changes, it needs to be updated in the other 3 documents, which can result in a complex query.
2. Possible increase in document sizes, especially for popular courses or prolific instructors

2. Data Migration(5 points)

Database Setup, Connections and Configurations:

1. Postgresql: I used the official PostgreSQL docker image and installed the PSQL command line interface to access the DB.
   Its connection is established using **psycogp2** library.
   Port : 5432
   User: admin
   Password : password
2. MongoDB: I used its official Docker image to run the server locally, and installed MongoDB Compass to visualize the data.
   Its connection is established using **pymongo** library.
   Port : 27017
   User: admin
   Password : password

DATA cleaning and ETL Pipeline

Throughout the process, numeric IDs are converted to strings for consistency in MongoDB. Existing data in MongoDB collections is cleared before migration to prevent conflicts.

1. **Extraction** Process
   a. Made a function **fetch_postgres_data** which establishes a connection to PostgreSQL, Creates a cursor using **RealDictCursor** to return the result of the query in form of Key value pairs, which is required for MongoDB schema.
   b. To Extract all Entities from Postegrsql data I implemented queries using Joins.

2. **Transformation** Process : For Each Table after Extracting the data from postegresql, I converted the numerical ids to string because they will be used as keys in Mongodb. I changed attribute name for proper naming convention and to avoid confusion.
   a. Departments Collection:
      i. Added empty arrays for courses, instructors, and students.
      ii. Added course_id, course_name
      iii. Added instructor_id , instructor_name
      iv. Added students_id , students_name
   b. Instructor Collection:
      i. Department ID replaced with department name.
      ii. Added empty array for courses taught(course_id, course_name).
   c. Students Collection:
      i. Department ID replaced with department name.
      ii. Added empty array for enrollments.
      iii. In the students ETL, department information is only included if it exists.(NULL values handled)

   d. Courses Collection:
      i. Department ID replaced with department name.
      ii. Added empty array for enrollments.
      iii. Added empty array for Instructors.(instructor_id , instructor_name)

3. **Load Process**
   After Transforming the data key-value pairs for Mongo, the collections are updated using Operations like **update_one**

   Entity Loading Order:
   - Departments
   - Instructors
   - Students
   - Courses
   - Enrollments

   Use of $addToset operator in MongoDB updates to prevent duplicate entries in arrays.

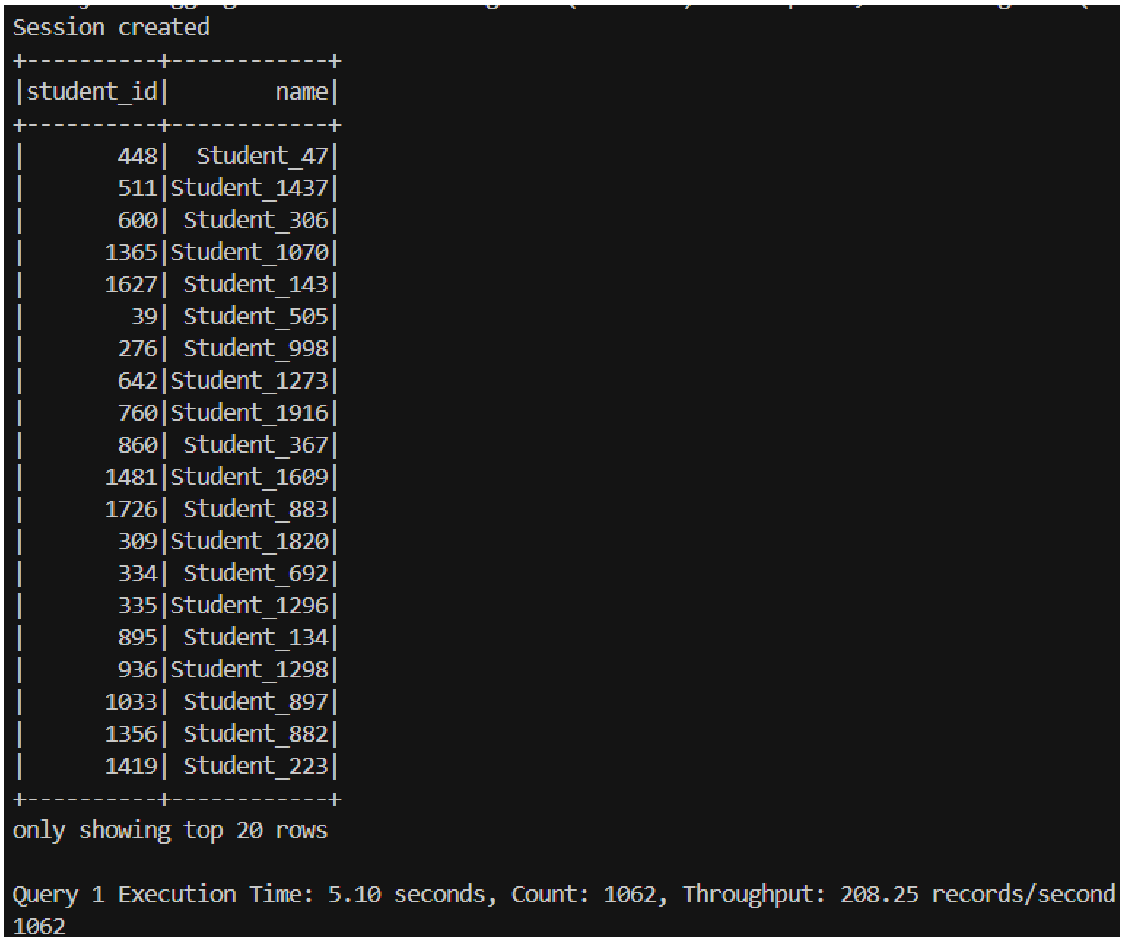
   This order ensures that all necessary reference documents exist before creating dependent documents. Initial documents are created using insert_one operations. Related documents are updated using update_one operations with the $push operator to add references.

3. Query Implementation using Apache Spark (10 points)

I found two methods to retrieve the results of a given query.

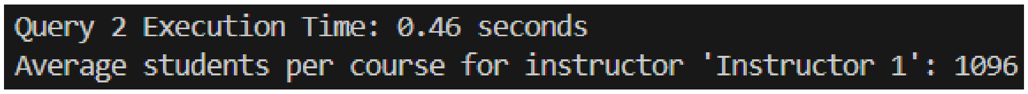
1. Apache Spark connector is used to convert the collection of MongoDB into a spark data frame, which is then manipulated to get the desired result.
   a. Advantages: If the data is small, it is beneficial to collect the entire dataset only once, store it locally, and query the DataFrame. This means Apache does not have to call the database repeatedly, reducing the network calls to the server.
   b. Disadvantages**:** When the data is extensive, we cannot just create a data frame for the entire dataset and store it locally. In such cases, a different technique is required, as this results in high memory consumption.
2. **AGGREGATION PIPELINE OF SPARK(one of the optimization techniques)**
   a. allows us to filter and process data directly within MongoDB
   b. This minimizes the amount of data that needs to be transferred to Spark, reducing network bandwidth usage and speeding up the overall process.

QUERY 1:



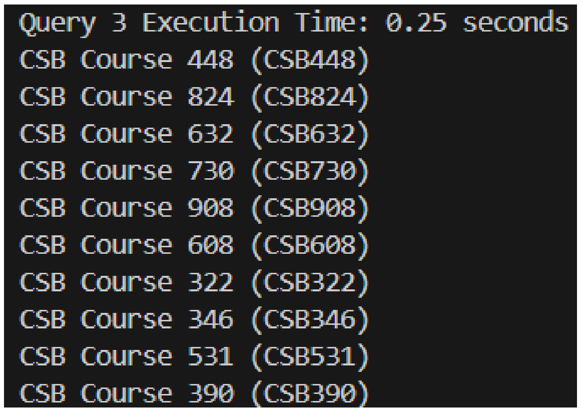
```
Session created
+----------+-----------+
|student_id|       name|
+----------+-----------+
|       448| Student_47|
|       511|Student_1437|
|       600| Student_306|
|      1365|Student_1070|
|      1627| Student_143|
|        39| Student_505|
|       276| Student_998|
|       642|Student_1273|
|       760|Student_1916|
|       860| Student_367|
|      1481|Student_1609|
|      1726| Student_883|
|       309|Student_1820|
|       334| Student_692|
|       335|Student_1296|
|       895| Student_134|
|       936|Student_1298|
|      1033| Student_897|
|      1356| Student_882|
|      1419| Student_223|
+----------+-----------+
only showing top 20 rows

Query 1 Execution Time: 5.10 seconds, Count: 1062, Throughput: 208.25 records/second
1062
```

QUERY 2:



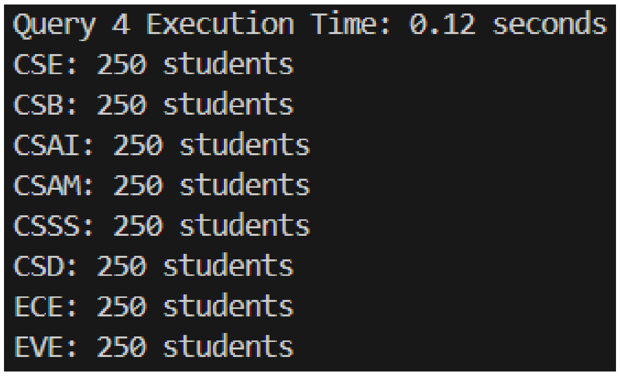
```
Query 2 Execution Time: 0.46 seconds
Average students per course for instructor 'Instructor 1': 1096
```

QUERY 3:



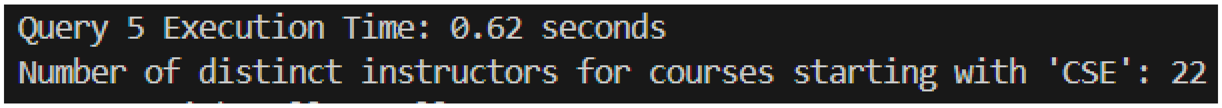
```
Query 3 Execution Time: 0.25 seconds
CSB Course 448 (CSB448)
CSB Course 824 (CSB824)
CSB Course 632 (CSB632)
CSB Course 730 (CSB730)
CSB Course 908 (CSB908)
CSB Course 608 (CSB608)
CSB Course 322 (CSB322)
CSB Course 346 (CSB346)
CSB Course 531 (CSB531)
CSB Course 390 (CSB390)
```

QUERY 4:



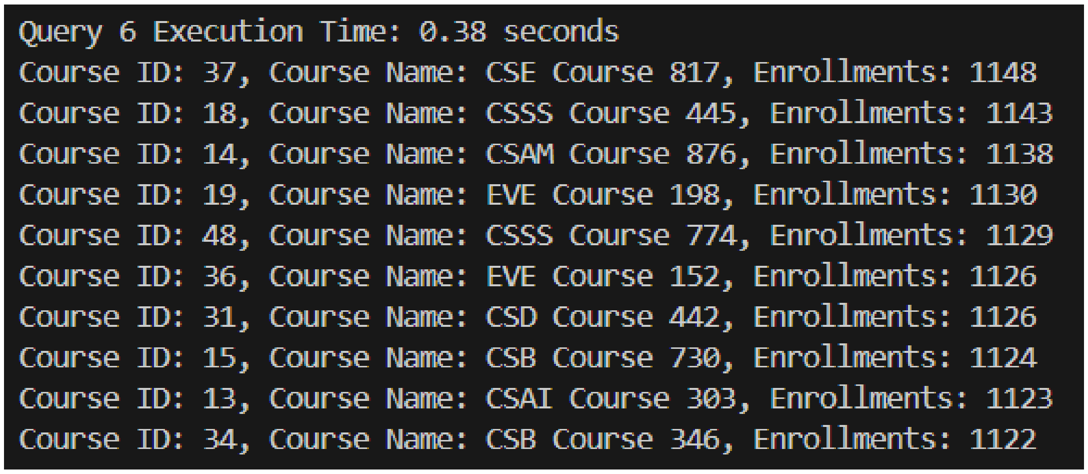
```
Query 4 Execution Time: 0.12 seconds
CSE: 250 students
CSB: 250 students
CSAI: 250 students
CSAM: 250 students
CSSS: 250 students
CSD: 250 students
ECE: 250 students
EVE: 250 students
```

QUERY 5:



```
Query 5 Execution Time: 0.62 seconds
Number of distinct instructors for courses starting with 'CSE': 22
```

QUERY 6:



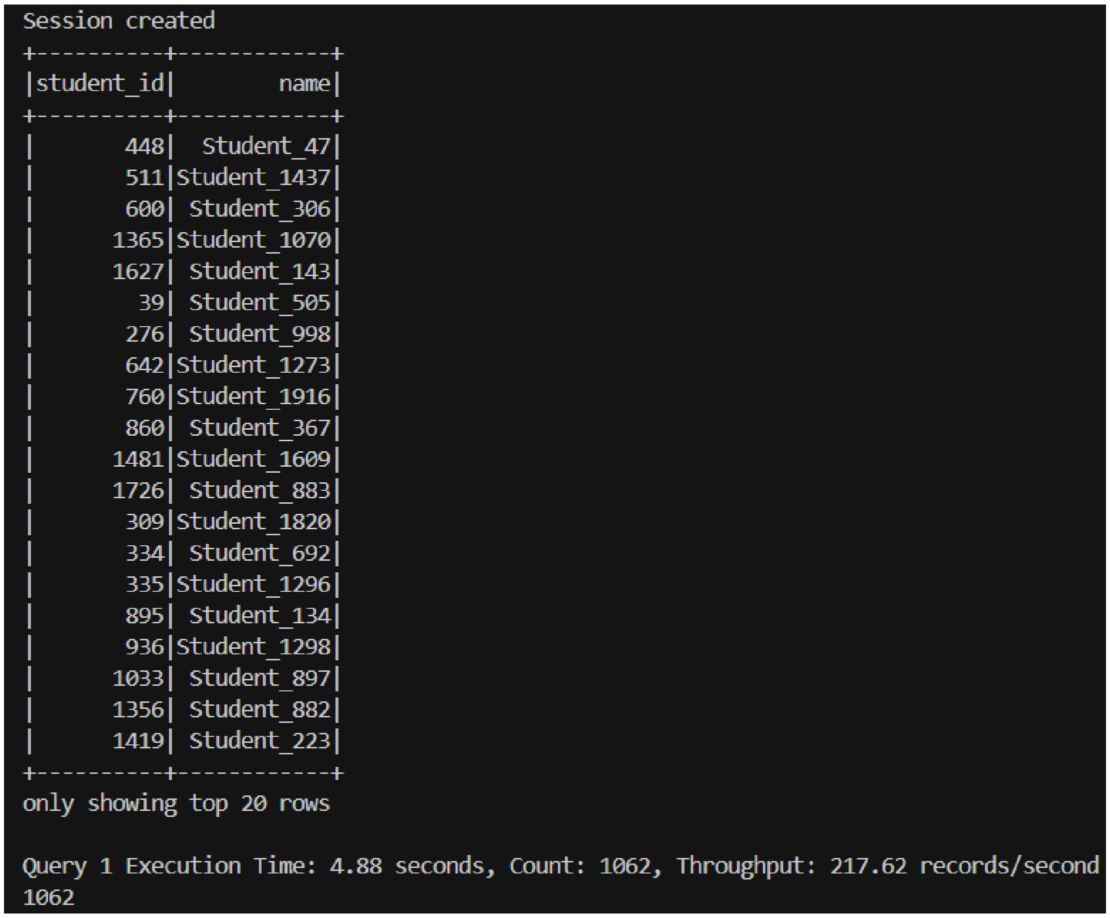
```
Query 6 Execution Time: 0.38 seconds
Course ID: 37, Course Name: CSE Course 817, Enrollments: 1148
Course ID: 18, Course Name: CSSS Course 445, Enrollments: 1143
Course ID: 14, Course Name: CSAM Course 876, Enrollments: 1138
Course ID: 19, Course Name: EVE Course 198, Enrollments: 1130
Course ID: 48, Course Name: CSSS Course 774, Enrollments: 1129
Course ID: 36, Course Name: EVE Course 152, Enrollments: 1126
Course ID: 31, Course Name: CSD Course 442, Enrollments: 1126
Course ID: 15, Course Name: CSB Course 730, Enrollments: 1124
Course ID: 13, Course Name: CSAI Course 303, Enrollments: 1123
Course ID: 34, Course Name: CSB Course 346, Enrollments: 1122
```

4. Performance Analysis and Optimization (5 points)

1. Indexing:

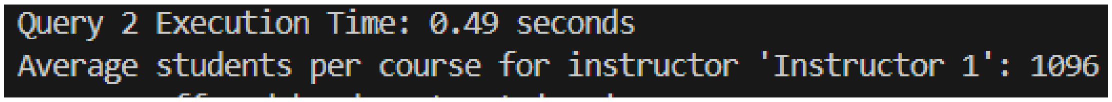
1. On Courses collection with course_id
2. On Students collection with student_id

QUERY 1: Time decreases after indexing and throughput increased slightly



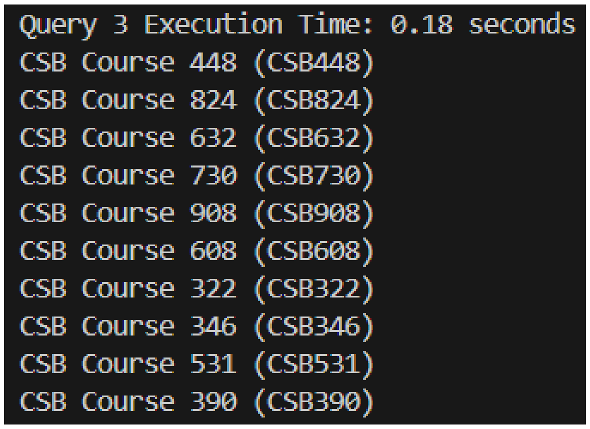
```
Session created
+----------+------------+
|student_id|        name|
+----------+------------+
|       448|  Student_47|
|       511|Student_1437|
|       600| Student_306|
|      1365|Student_1070|
|      1627| Student_143|
|        39| Student_505|
|       276| Student_998|
|       642|Student_1273|
|       760|Student_1916|
|       860| Student_367|
|      1481|Student_1609|
|      1726| Student_883|
|       309|Student_1820|
|       334| Student_692|
|       335|Student_1296|
|       895| Student_134|
|       936|Student_1298|
|      1033| Student_897|
|      1356| Student_882|
|      1419| Student_223|
+----------+------------+
only showing top 20 rows

Query 1 Execution Time: 4.88 seconds, Count: 1062, Throughput: 217.62 records/second
1062
```

QUERY 2: Time increases after indexing.



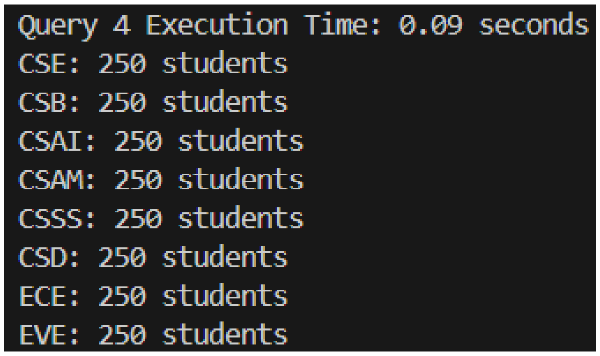
```
Query 2 Execution Time: 0.49 seconds
Average students per course for instructor 'Instructor 1': 1096
```

QUERY 3: Time decreases after indexing



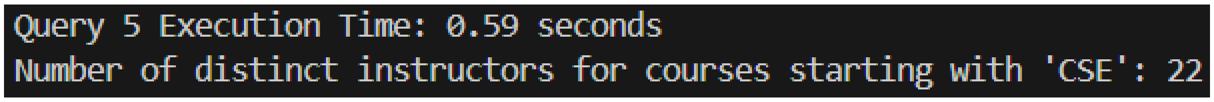
```
Query 3 Execution Time: 0.18 seconds
CSB Course 448 (CSB448)
CSB Course 824 (CSB824)
CSB Course 632 (CSB632)
CSB Course 730 (CSB730)
CSB Course 908 (CSB908)
CSB Course 608 (CSB608)
CSB Course 322 (CSB322)
CSB Course 346 (CSB346)
CSB Course 531 (CSB531)
CSB Course 390 (CSB390)
```

QUERY 4: Time decreases after indexing



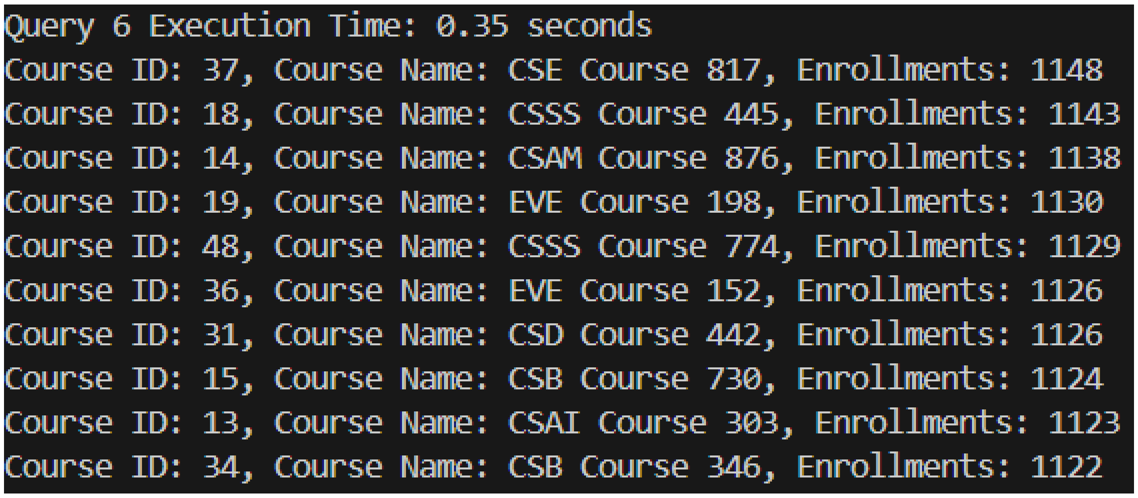
```
Query 4 Execution Time: 0.09 seconds
CSE: 250 students
CSB: 250 students
CSAI: 250 students
CSAM: 250 students
CSSS: 250 students
CSD: 250 students
ECE: 250 students
EVE: 250 students
```

QUERY 5: Time decreases after indexing



```
Query 5 Execution Time: 0.59 seconds
Number of distinct instructors for courses starting with 'CSE': 22
```

QUERY 6: Time decreases after indexing

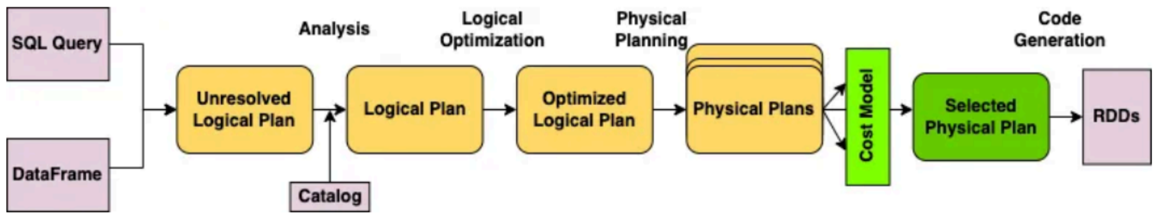


```
Query 6 Execution Time: 0.35 seconds
Course ID: 37, Course Name: CSE Course 817, Enrollments: 1148
Course ID: 18, Course Name: CSSS Course 445, Enrollments: 1143
Course ID: 14, Course Name: CSAM Course 876, Enrollments: 1138
Course ID: 19, Course Name: EVE Course 198, Enrollments: 1130
Course ID: 48, Course Name: CSSS Course 774, Enrollments: 1129
Course ID: 36, Course Name: EVE Course 152, Enrollments: 1126
Course ID: 31, Course Name: CSD Course 442, Enrollments: 1126
Course ID: 15, Course Name: CSB Course 730, Enrollments: 1124
Course ID: 13, Course Name: CSAI Course 303, Enrollments: 1123
Course ID: 34, Course Name: CSB Course 346, Enrollments: 1122
```

Results:

1. Since there were fewer courses, the indexing was not necessary. Besides, it increases the time for execution of queries because it requires the throughput for indexing data structure..
2. Now, the execution time for queries 1, 2 and 4 decreases slightly because of indexing. With 2000 students, searches for student records are more time-consuming without an index.

2. **Adaptive Query Execution (AQE)** Optimizer in Apache Spark is an advanced optimization engine that dynamically tunes the execution plan for a given query, based on the runtime statistics of the data. It can improve query performance by optimizing the data processing pipelines.

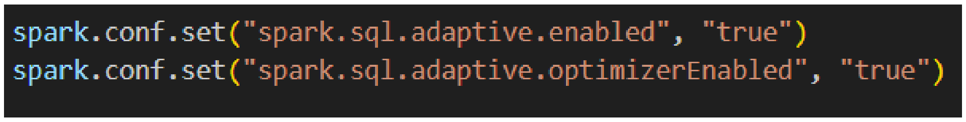


Spark SQL Catalyst AQE Stages

The AQE optimizer in Spark provides the following benefits:

1. Dynamic optimization
2. Adaptive planning
3. Cost-based optimization
4. Reduced overhead
5. Faster processing

Implementation:

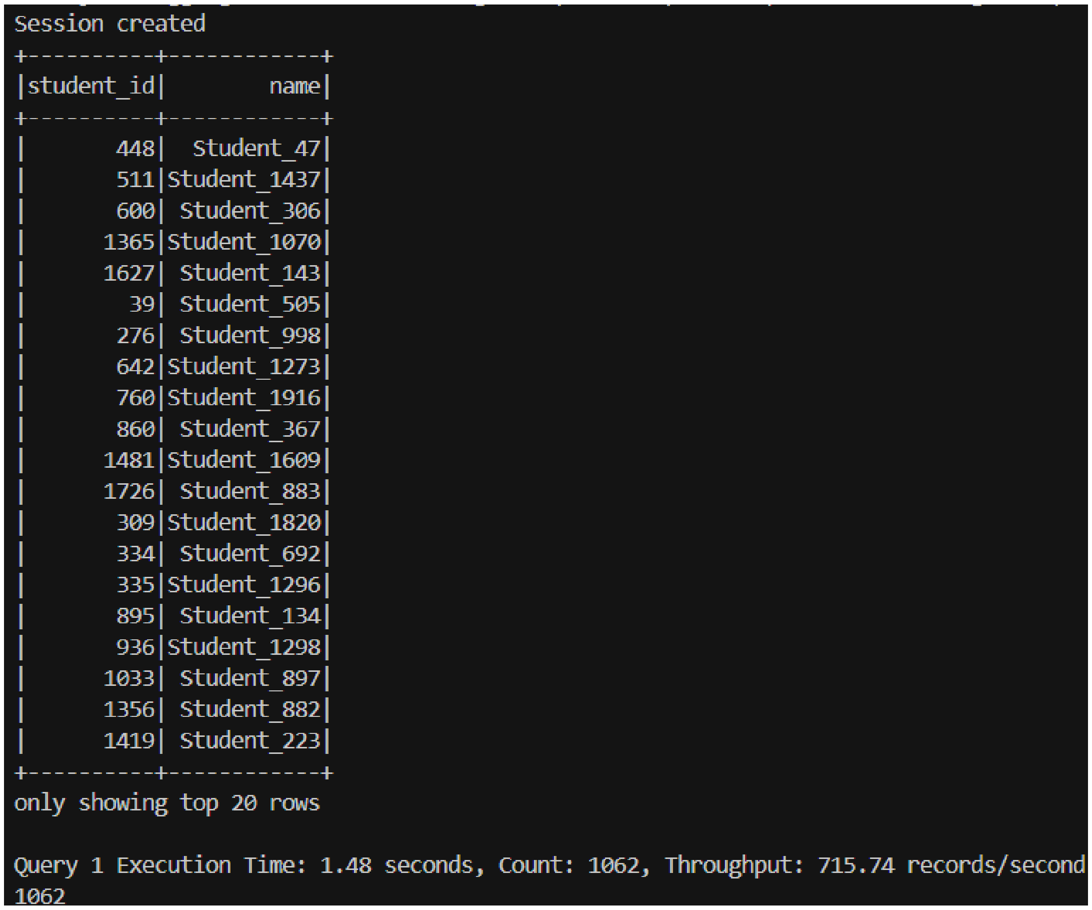


```
spark.conf.set("spark.sql.adaptive.enabled", "true")
spark.conf.set("spark.sql.adaptive.optimizerEnabled", "true")
```

Impact:

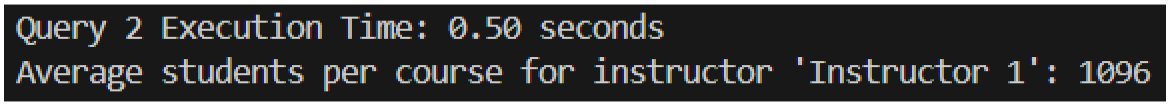
1. When this Configurations are enabled, Spark can dynamically chose and optimize joins and aggregations during the execution process.
2. It allows Spark to leverage techniques like coalescing partitions, which can improve the efficiency of data processing and reduce shuffling costs.
3. AQE can minimize the amount of data shuffled between partitions in distributed systems.
4. We can see the result for time execution of each query is reduced.
    a. For 1st query there is approx 70% improvement in query performance.(4secs to 1sec)
    b. Though the data is not big enough to properly the justify the changes in the time of other queries, it will create a huge effect when it comes to complex queries.

QUERY 1: Time decreases significantly after indexing + Adaptive Query Execution
We can also see the throuput increased.



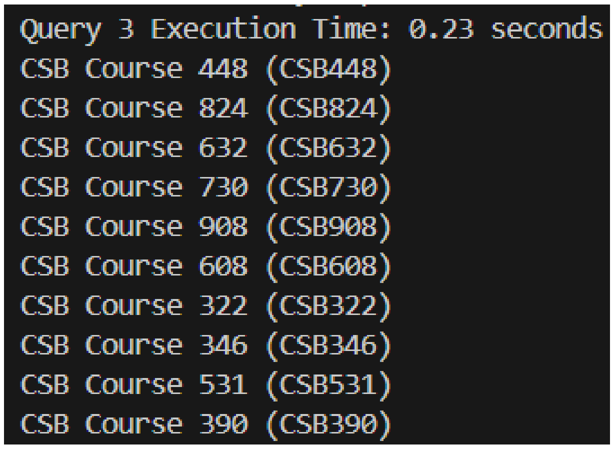
```
Session created
+----------+------------+
|student_id|        name|
+----------+------------+
|       448|  Student_47|
|       511|Student_1437|
|       600| Student_306|
|      1365|Student_1070|
|      1627| Student_143|
|        39| Student_505|
|       276| Student_998|
|       642|Student_1273|
|       760|Student_1916|
|       860| Student_367|
|      1481|Student_1609|
|      1726| Student_883|
|       309|Student_1820|
|       334| Student_692|
|       335|Student_1296|
|       895| Student_134|
|       936|Student_1298|
|      1033| Student_897|
|      1356| Student_882|
|      1419| Student_223|
+----------+------------+
only showing top 20 rows

Query 1 Execution Time: 1.48 seconds, Count: 1062, Throughput: 715.74 records/second
1062
```

QUERY 2: Time was approx same.

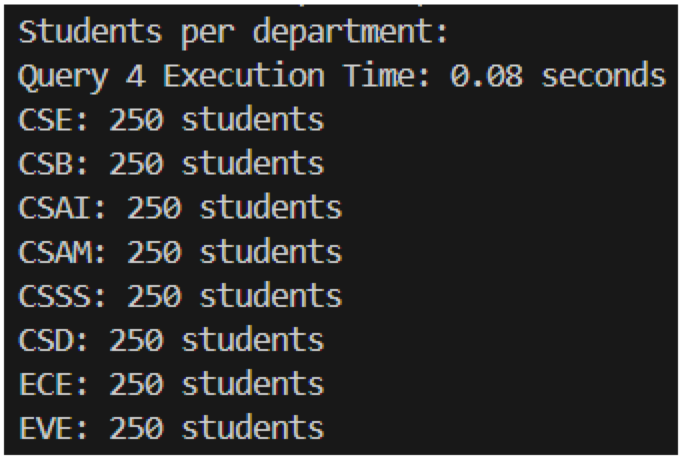


```
Query 2 Execution Time: 0.50 seconds
Average students per course for instructor 'Instructor 1': 1096
```

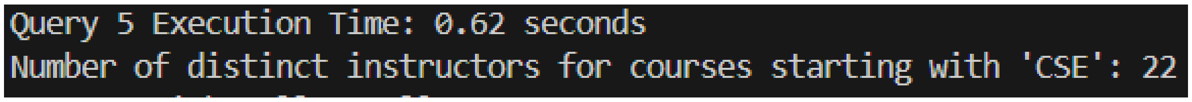
QUERY 3: Time increased


```
Query 3 Execution Time: 0.23 seconds
CSB Course 448 (CSB448)
CSB Course 824 (CSB824)
CSB Course 632 (CSB632)
CSB Course 730 (CSB730)
CSB Course 908 (CSB908)
CSB Course 608 (CSB608)
CSB Course 322 (CSB322)
CSB Course 346 (CSB346)
CSB Course 531 (CSB531)
CSB Course 390 (CSB390)
```

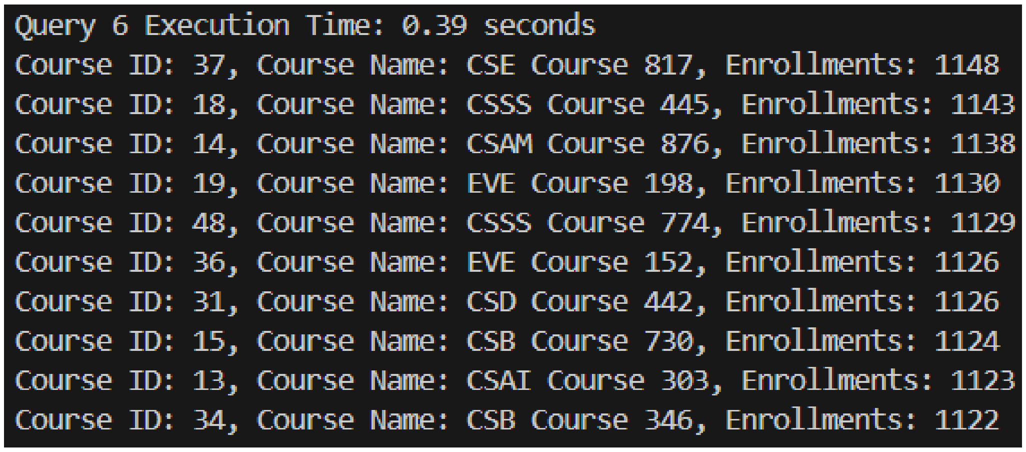
QUERY 4: Time slightly decreased


```
Students per department:
Query 4 Execution Time: 0.08 seconds
CSE: 250 students
CSB: 250 students
CSAI: 250 students
CSAM: 250 students
CSSS: 250 students
CSD: 250 students
ECE: 250 students
EVE: 250 students
```

QUERY 5: Time increased


```
Query 5 Execution Time: 0.62 seconds
Number of distinct instructors for courses starting with 'CSE': 22
```

QUERY 6: Time increased


```
Query 6 Execution Time: 0.39 seconds
Course ID: 37, Course Name: CSE Course 817, Enrollments: 1148
Course ID: 18, Course Name: CSSS Course 445, Enrollments: 1143
Course ID: 14, Course Name: CSAM Course 876, Enrollments: 1138
Course ID: 19, Course Name: EVE Course 198, Enrollments: 1130
Course ID: 48, Course Name: CSSS Course 774, Enrollments: 1129
Course ID: 36, Course Name: EVE Course 152, Enrollments: 1126
Course ID: 31, Course Name: CSD Course 442, Enrollments: 1126
Course ID: 15, Course Name: CSB Course 730, Enrollments: 1124
Course ID: 13, Course Name: CSAI Course 303, Enrollments: 1123
Course ID: 34, Course Name: CSB Course 346, Enrollments: 1122
```