Big Data Analytics

(Assignment-1 on Mongo DB)

Data Modeling and Schema Design

Postgres-SQL:

Database name : **institute**

Tables:

1. **courses**

   Fields:

   > course_id SERIAL PRIMARY KEY,
   > course_name VARCHAR(255) NOT NULL,
   > department_id INT NOT NULL,   -- Links to Departments table
   > instructor_id INT NOT NULL,   -- Links to Instructors table
   > is_core BOOLEAN NOT NULL,   -- True if the course is core, False otherwise
   > FOREIGN KEY (department_id) REFERENCES Departments(department_id),
   > FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id)

2. **departments**

   Fields:

   > department_id SERIAL PRIMARY KEY,
   > department_name VARCHAR(255) NOT NULL

3. **enrollments**

   Fields: enrollment_id, student_id, course_id, enrollment_date, semester

   > enrollment_id SERIAL PRIMARY KEY,
   > student_id INT NOT NULL,   -- Links to Students table
   > course_id INT NOT NULL,   -- Links to Courses table
   > enrollment_date DATE,
   > semester VARCHAR(50),

> FOREIGN KEY (student_id) REFERENCES Students(student_id),
> FOREIGN KEY (course_id) REFERENCES Courses(course_id),

4. **instructors**

   Fields:

   > instructor_id SERIAL PRIMARY KEY,
   > name VARCHAR(255) NOT NULL,
   > department_id INT NOT NULL,   -- Links to Departments table
   > FOREIGN KEY (department_id) REFERENCES Departments(department_id)

5. **students**

   Fields:

   > student_id SERIAL PRIMARY KEY,
   > name VARCHAR(255) NOT NULL,
   > department_id INT NOT NULL,
   > FOREIGN KEY (department_id) REFERENCES Departments(department_id)

Mongo DB Schema:

Departments:

```
{
    "_id": department_id,
    "department_name": "String", // Name of the department (e.g., "Computer Science")
    "instructors": [             // Embedded list of instructors in the department
        {
        "instructor_id": ObjectId(),  // Reference to the instructor's _id
        "name": "String"              // Name of the instructor
        }
    ],
    "students": [                // Embedded list of students in the department
        {
        "student_id": ObjectId(),  // Reference to the student's _id
        "name": "String"           // Name of the student
        }
    ]
}
```

Instructors:

```
{
    "_id": instructor_id,
    "name": "String",            // Name of the instructor
    "department": ObjectId(),    // Reference to the _id of the Departments collection
    "courses": [                 // Embedded list of courses taught by the instructor
        {
        "course_id": ObjectId(), // Reference to the course _id from the Courses collection
        "is_core": Boolean       // Indicates if the course is a core course
        }
    ]
}
```

Courses:

```
{
    "_id": course_id,
    "course_name": "String",     // Name of the course
    "department": ObjectId(),    // Reference to the _id of the Departments collection
    "instructor": ObjectId(),    // Reference to the _id of the Instructors collection
    "is_core": Boolean,          // True if the course is a core course, False otherwise
    "enrollments": [             // Embedded list of students enrolled in the course
        {
        "student_id": ObjectId(),    // Reference to the student's _id from Students collection
        "student_name": "String",    // Name of the student
        "enrollment_date": Date,     // Date of enrollment
        "semester": "String"         // Semester of enrollment
        }
    ]
}
```

Students:

```
{
    "_id": student_id,
    "name": "String",           // Name of the student
    "department": ObjectId(),    // Reference to the _id of the Departments collection
    "enrollments": [            // Embedded list of courses the student is enrolled in
        {
        "course_id": ObjectId(),     // Reference to the course _id from the Courses collection
        "enrollment_date": Date,     // Date of enrollment
        "semester": "String"        // Semester of enrollment
        }
    ]
}
```

Mapping Description:

1. Departments Table → Departments Collection

- **Mapping**: The `departments` table is mapped to the **Departments** collection, with embedded lists of **instructors** and **students**.
- **Denormalization**: Embedding **instructors** and **students** within the department reduces the need for separate collections or complex joins when retrieving department-related data, and improved performance for queries like fetching instructors or students of a specific department.

2. Instructors Table → Instructors Collection

- **Mapping**: The `instructors` table is mapped to the **Instructors** collection, with an embedded array of **courses** taught by each instructor.
- **Denormalization**: Embedding courses simplifies access to all courses taught by an instructor in one query, improving read efficiency.

3. Courses Table → Courses Collection

- **Mapping**: The `courses` table is mapped to the **Courses** collection, with **enrollments** (students) embedded in the course document.
- **Denormalization**: Embedding students directly within the course document improves performance by simplifying the process of retrieving all students enrolled in a course.

4. Students Table → Students Collection

- **Mapping**: The `students` table is mapped to the **Students** collection, with an embedded array of **enrollments** (courses).
- **Denormalization**: Embedding courses within student documents ensures quick access to enrollment details, removing the need for a separate enrollment table.

5. Enrollments Table → Embedded in Courses and Students Collections

- **Mapping**: The `enrollments` table is denormalized and embedded within both the **Courses** and **Students** collections.
- **Justification**: Embedding simplifies access to enrollment data from either the student's or the course's perspective, enhancing performance for queries like fetching details of all the students enrolled in a specific course, and fetching details of all courses enrolled by a specific student.

Data Migration:

Steps:

1. Extracted all data from Postgres-SQL schema, stored the data extracted for each table into different lists.
2. As, I have populated the data myself there were no missing values and hence no data cleaning is required.
3. Transform the relational schema into a MongoDB-compatible document structure. This involves embedding related data, for example storing the enrollment details of students in courses collection, and storing the enrollment details of courses in students collection.
4. Finally after storing the migrated data, loaded the data into Mongo DB using '**insert_many**'.

Query Implementation using Apache Spark:

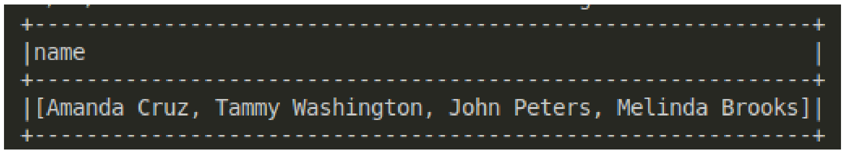
1. Fetching all students enrolled in a specific course.

   All students enrolled in Machine Learning course:

   query:

```
def query1():
    machine_learning_students = courses_df \
        .filter(courses_df.course_name == "Machine Learning") \
        .select("enrollments.name")

    machine_learning_students.show(truncate=False)
```



```
+----------------------------------------------------------+
|name                                                      |
+----------------------------------------------------------+
|[Amanda Cruz, Tammy Washington, John Peters, Melinda Brooks]|
+----------------------------------------------------------+
```

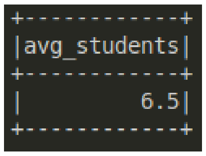
2. Calculating the average number of students enrolled in courses offered by a particular instructor at the university.

   Average number of students enrolled in courses taught by an instructor with ID 31:

   query:

```
def query2():
    instructor_id = 31
    instructor_courses = courses_df.filter(courses_df.instructor == instruct

    avg_students = instructor_courses \
        .withColumn("num_students", F.size("enrollments")) \
        .agg(F.avg("num_students").alias("avg_students"))
    avg_students.show()
```



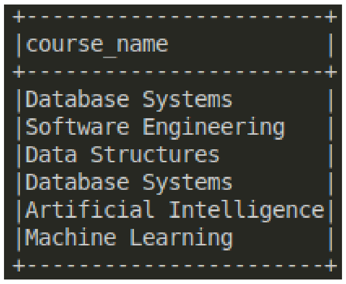
```
+------------+
|avg_students|
+------------+
|         6.5|
+------------+
```

3. Listing all courses offered by a specific department.

   All courses offered by the department with ID 1 (CSE department):

```
def query3():
    cse_courses = courses_df.filter(courses_df.department == 1).select("cour

    cse_courses.show(truncate=False)
```



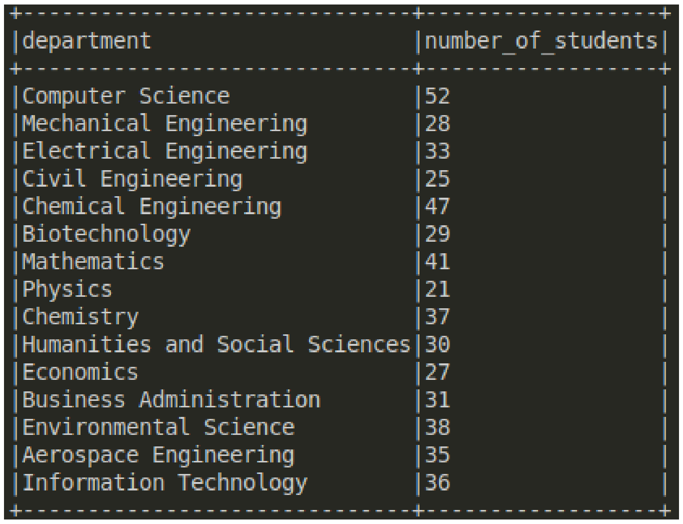
```
+-----------------------+
|course_name            |
+-----------------------+
|Database Systems       |
|Software Engineering   |
|Data Structures        |
|Database Systems       |
|Artificial Intelligence|
|Machine Learning       |
+-----------------------+
```

4. Finding the total number of students per department.

   query:

```
def query4():
    total_students_per_dept = departments_df \
        .withColumn("number_of_students", F.size("students")) \
        .select(F.col("department_name").alias("department"), "number_of_stu

    total_students_per_dept.show(truncate=False)
```

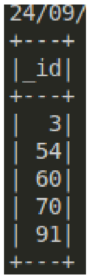


```
+----------------------------+------------------+
|department                  |number_of_students|
+----------------------------+------------------+
|Computer Science            |52                |
|Mechanical Engineering      |28                |
|Electrical Engineering      |33                |
|Civil Engineering           |25                |
|Chemical Engineering        |47                |
|Biotechnology               |29                |
|Mathematics                 |41                |
|Physics                     |21                |
|Chemistry                   |37                |
|Humanities and Social Sciences|30              |
|Economics                   |27                |
|Business Administration     |31                |
|Environmental Science       |38                |
|Aerospace Engineering       |35                |
|Information Technology      |36                |
+----------------------------+------------------+
```

5. Finding instructors who have taught all the BTech CSE core courses sometime during their tenure at the university.

   a. list of all cse core courses

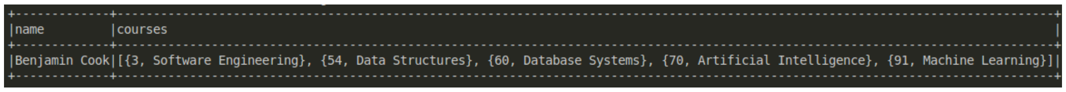
```
cse_core_courses = courses_df.filter(
        (courses_df.department == 1) & (courses_df.is_core == True)
    ).select("_id")
```



```
24/09/
+---+
|_id|
+---+
|  3|
| 54|
| 60|
| 70|
| 91|
+---+
```

   b. instructor who taught all these cse core courses:

```
core_course_ids = [row['_id'] for row in cse_core_courses.collect()]
    instructors_taught_all = instructors_df.filter(
        F.array_contains(F.col("courses._id"), core_course_ids[0])
    )
    for course_id in core_course_ids[1:]:
        instructors_taught_all = instructors_taught_all.filter(
            F.array_contains(F.col("courses._id"), course_id)
        )
    instructors_taught_all.select("name", "courses").show(truncate=False
```



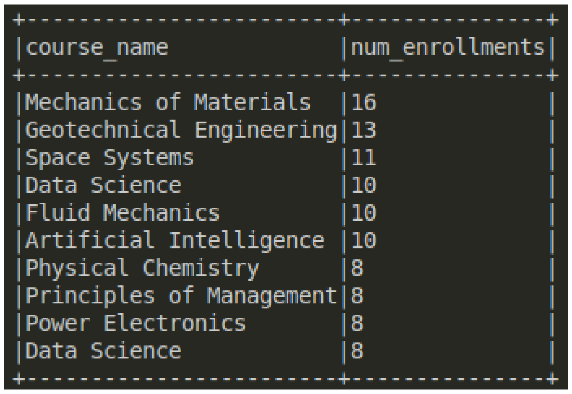
```
+-------------+----------------------------------------------------------------------------------------------------------+
|name         |courses                                                                                                   |
+-------------+----------------------------------------------------------------------------------------------------------+
|Benjamin Cook|[{3, Software Engineering}, {54, Data Structures}, {60, Database Systems}, {70, Artificial Intelligence}, {91, Machine Learning}]|
+-------------+----------------------------------------------------------------------------------------------------------+
```

6. Finding top-10 courses with the highest enrollments.

```
courses_with_enrollments = courses_df.withColumn(
    "num_enrollments", F.size("enrollments")
)

top_10_courses = courses_with_enrollments.orderBy(F.desc("num_enrollments")

top_10_courses.select("course_name", "num_enrollments").show(truncate=False
```



```
+-----------------------+---------------+
|course_name            |num_enrollments|
+-----------------------+---------------+
|Mechanics of Materials |16             |
|Geotechnical Engineering|13            |
|Space Systems          |11             |
|Data Science           |10             |
|Fluid Mechanics        |10             |
|Artificial Intelligence|10             |
|Physical Chemistry     |8              |
|Principles of Management|8             |
|Power Electronics      |8              |
|Data Science           |8              |
+-----------------------+---------------+
```

Performance based on execution time:

Query	No optimization
1	0.1705358028411865
2	0.1102173328399658
3	0.0147087574005126
4	0.0256495475769042
5	0.1874008178710937
6	0.0290093421936035

Performance Analysis and Optimization

Index creation:

`db.courses.createIndex({ department: 1, is_core: 1 });`

`db.courses.createIndex({ "enrollments._id": 1 });`

`db.courses.createIndex({ instructor: 1 });`

`db.instructors.createIndex({ "courses._id": 1 });`

`db.departments.createIndex({ "students._id": 1 });`

Partition creation:

```
courses_df = courses_df.repartition(10, "department", "_id")
departments_df = departments_df.repartition(5,"_id")
instructors_df = instructors_df.repartition(7,"_id")
```

- `courses_df` **partitioned by** `department` **and** `_id`:
  - `department`: Optimizes queries filtering by department, such as listing courses by department or finding core courses.
  - `_id`: Ensures efficient handling of course-specific queries (e.g., enrollments, course details).
- `departments_df` **partitioned by** `department_id`:
  - `department_id`: Ensures faster access to department-related data and avoids shuffling for department-based queries.
- `instructors_df` **partitioned by** `instructor_id`:
  - `instructor_id`: Improves performance of queries that filter or group by instructors (e.g., courses taught by a specific instructor).

Performance comparison:

Query	No optimization	Indexing	Partioning	Indexing + Partitioning
1	0.1705358028411865	0.1134316921234130	0.141143560409545	0.16926145553588867
2	0.1102173328399658	0.1266441345214843	0.066459655761718	0.06744527816772461
3	0.0147087574005126	0.0139634609222412	0.012083292007446	0.01235222816467285
4	0.0256495475769042	0.0241308212280273	0.020113229751586	0.02397704124450683
5	0.1874008178710937	0.1915709972381591	0.211130142211914	0.31534838676452637
6	0.0290093421936035	0.0253927707672119	0.024239301681518	0.02047514915466308

There is no fixed pattern for recognizing any optimization because we have a very small database with very less number of entries.

However, **Indexing + Partitioning** performs better than **No optimizatio**n in 5/6 queries.

Indexing performs better than **No optimization** in 3/6 queries.

Only **Partitioning** performs better than **No optimization** in 5/6 queries.

- In small databases, the amount of data is typically small enough to fit within a single node's memory, meaning the overhead of shuffling data between nodes (which indexing and partitioning aim to minimize) is negligible.
- The performance gain from indexing or partitioning becomes marginal since most operations can be completed quickly without needing complex optimization techniques.
- For small databases, loading the index may take more time than actually executing the query with utilisation of index hence will become an overhead and will lead to increased execution time.