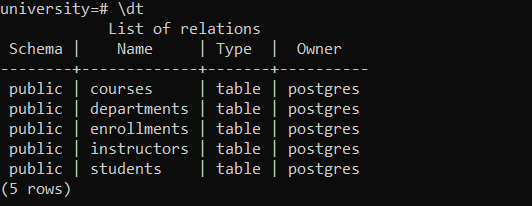
**BDA ASSIGNMENT-1**

**Bhavya Gupta**

**2021245**

**PostgreSQL Schema**

****

**- Students Table:**

- student\_id: Unique identifier for each student (Primary Key).

- name: The student's name.

- age`: The student's age.

- departmentid: Foreign key referencing the `Departments` table.

**- Departments Table:**

- departmentid: Unique identifier for each department (Primary Key).

- name: The name of the department (e.g., Computer Science, Mathematics).

**- Instructors Table:**

instructor\_id: Unique identifier for each instructor (Primary Key).

name: The instructor's name.

department\_id: Foreign key referencing the `Departments` table.

**- Courses Table:**

courseid: Unique identifier for each course (Primary Key).

name: The course name

instructorid: Foreign key referencing the `Instructors` table.

departmentid: Foreign key referencing the `Departments` table.

coursetype: Whether it is core or elective

**- Enrollments Table:**

enrollmentid:unique identifier for each enrollment record (the primary key).

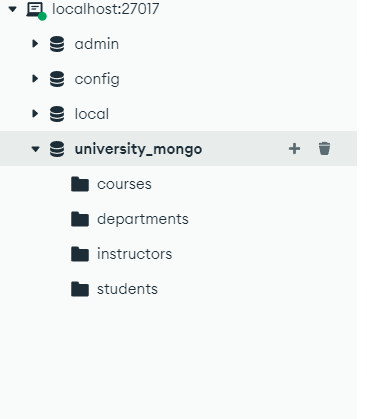
studentid:unique identifier of a student (foreign key referring to the Students table). courseid:unique identifier of a course (foreign key referring to the Courses table). enrollmentdate:This column records the date when the student enrolled in the course.

**Mapping to MongoDB Schema**

In MongoDB, aim is for a denormalized structure that improves performance by embedding related data within a single document.

**The migration process involves embedding (embedding in MongoDB means putting related information inside the same document rather than keeping it in separate collections) data where necessary, avoiding the need for multiple collections and joins, while ensuring that the system maintains the same integrity and accessibility as in the original RDBMS.**

Here’s how the PostgreSQL schema will map to MongoDB collections:

****

**1. Students Collection**

The students collection stores personal information about each student. This includes their ID, name, age, and department information. This structure provides a straightforward and efficient way to retrieve details about students, such as their department association, without the need for complex joins.

**2. Departments Collection**

The Departments collection stores data about university departments, such as the department ID and name.

**3. Instructors Collection**

The Instructors Collection in MongoDB stores details about instructors in a university system. Each document in this collection represents a single instructor and includes their basic personal and department information.

**4. Courses Collection**

Course Details: Information about the course, such as its ID (CourseID), name, department (DepartmentID and DepartmentName), instructor (InstructorID and InstructorName), and the type of course (CourseType).

Enrolled Students: The EnrolledStudents array holds detailed information about each student enrolled in the course. This includes:

* Student Personal Details: StudentID, Name, Age
* Student Department: DepartmentID, DepartmentName
* Enrollment Date: The date when each student enrolled in the course.

Total Enrollment Count: The total number of students enrolled in the course is stored in the TotalEnrollments field.

### **Denormalization Strategy**

**The denormalization in MongoDB reduces the need for costly joins by storing frequently queried data in a single location. Key areas of embedding include:**

**Courses Table:**

* **In PostgreSQL, Courses references Departments and Instructors via foreign keys (departmentid, instructorid).**
* **In MongoDB, course details, along with Department Name and Instructor Name, are embedded directly into the course document. This eliminates the need for joins.**

**Enrollments Table:**

* **In PostgreSQL, the Enrollments table tracks which students are enrolled in which courses.**
* **In MongoDB, enrollment details are embedded inside the course document under an array (EnrolledStudents), which includes each student's information, department, and enrollment date.**

**Instructors Table:**

* **In PostgreSQL, Courses references instructors via instructorid.**
* **In MongoDB, the Instructor Name and ID are embedded directly in the course document.**

### **2) MIGRATION**

**EXTRACTION**

**In this phase, data was extracted from the PostgreSQL database using SQL queries. The key tables extracted from the RDBMS. Each table was queried to retrieve all records from PostgreSQL. The extracted data was stored temporarily in memory for transformation.**

**TRANSFORMATION**

**In transformation phase, the extracted data was restructured to fit the document-based model of MongoDB. It involved**

**Embedding Data which is explain above**

**Data cleaning, some fields required formatting changes, such as converting PostgreSQL's date format into a string format compatible with MongoDB.**

**Field renaming and restructuring**

**LOADING**

**Once the data was transformed into the MongoDB-friendly document model, it was inserted into the respective MongoDB collections.The transformed data was inserted using MongoDB’s insert\_one() or insert\_many() methods for each collection.**

**A Python-based ETL script that connects to PostgreSQL, extracts and transforms the data, and loads it into MongoDB. The script handles each entity (e.g., students, courses) independently and ensures the integrity of embedded relationships.**

**3)Query Implementation Using Apache Spark**

* **The environment was set up using the MongoDB Spark connector, allowing Spark to interact directly with the MongoDB database.**

**Query 1: Fetching all students enrolled in a specific course**

**Purpose:**

- To retrieve a list of students (names and ages) enrolled in a particular course identified by a given `course\_id`.

**Code:**

course\_id=7

students\_enrolled\_in\_course = courses\_df \

.filter(col("CourseID") == course\_id) \

.select("EnrolledStudents.Name", "EnrolledStudents.Age")

students\_enrolled\_in\_course.show()

**Result**:list of students (name and age) who are enrolled in the specified course.

**Query 2: Calculating the average number of students enrolled in courses offered by a particular instructor\*\***

**Purpose:**

- To calculate the average number of students enrolled in all courses taught by a specific instructor identified by `instructor\_id`.

**Code:**

courses\_with\_enrollment\_count = courses\_by\_instructor \

.withColumn("num\_students", size(col("EnrolledStudents"))) \

.select("CourseID", "num\_students")

avg\_students\_per\_course = courses\_with\_enrollment\_count.agg(avg("num\_students").alias("avg\_students")).show()

**Result:**outputs the average number of students enrolled in courses taught by the given instructor.

**Query 3: Listing all courses offered by a specific department\*\***

**Purpose:**

- To retrieve a list of all courses offered by a specific department, identified by `department\_id`.

**Code:**

courses\_in\_department = courses\_df.filter(col("DepartmentID") == department\_id).select("Name")

courses\_in\_department.show()

**Result:**list of course names that belong to the specified department.

**Query 4: Finding the total number of students per department**

**Purpose:-** To calculate how many unique students are enrolled in each department.

**Code:**

students\_by\_department = students\_df.groupBy("DepartmentID").agg(countDistinct("StudentID").alias("total\_students"))

students\_by\_department.show()

**```**

**Result:**outputs a list of department IDs along with the corresponding total number of students in each department.

**Query 5: Finding instructors who have taught all the BTech CSE core courses**

**Purpose-** To identify instructors who have taught every core course in the Computer Science department (BTech CSE).

**Code:**

core\_courses = courses\_df.filter((col("CourseType") == core\_course\_type) & (col("DepartmentName") == department\_name))

instructors\_who\_taught\_all\_core\_courses = instructors\_df \

.join(core\_courses, instructors\_df.InstructorID == core\_courses.InstructorID, "inner") \

.groupBy(instructors\_df.InstructorID, instructors\_df.Name) \

.agg(countDistinct(core\_courses.CourseID).alias("num\_core\_courses")) \

.filter(col("num\_core\_courses") == core\_courses.count())

instructors\_who\_taught\_all\_core\_courses.show()

**Result:**a list of instructors who have taught every core course in the Computer Science department.

**Query 6: Finding top-10 courses with the highest enrollments**

**Purpose:**To find the top 10 courses with the highest number of enrolled students.

**Code:**

top\_10\_courses\_by\_enrollment = courses\_df \

.withColumn("num\_students", size(col("EnrolledStudents"))) \

.select("CourseID", "Name", "num\_students") \

.orderBy(col("num\_students").desc()) \

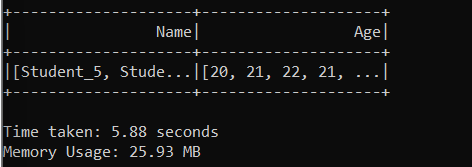
.limit(10)

top\_10\_courses\_by\_enrollment.show()

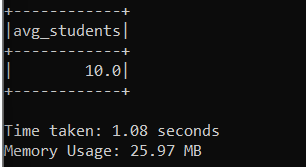
**Result**:a list of the top 10 courses with the most students enrolled, along with the number of students in each course.

**Query Results and Performance Observations**

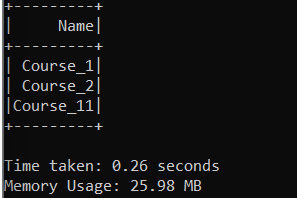
**Query 1: Fetching All Students Enrolled in a Specific Course**

****

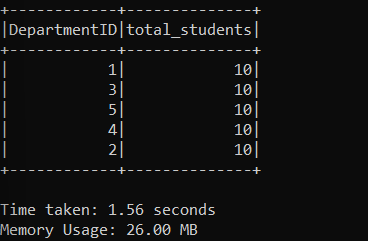
**Query 2: Calculating the Average Number of Students per Course for a Specific Instructor**

****

**Query 3: Listing All Courses Offered by a Specific Department**

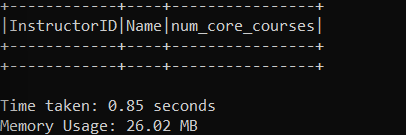
****

**Query 4: Finding the Total Number of Students per Department**

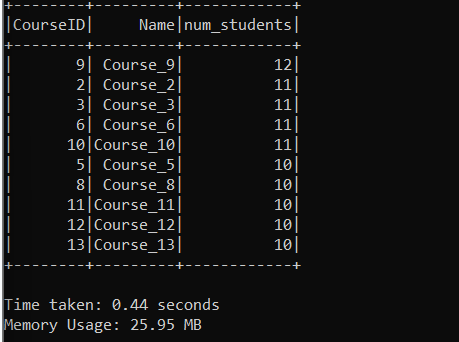
****

**Query 5: Listing Instructors Who Have Taught All Core Courses in BTech CSE**

No instructor teaches all the core courses of CSE department. As my data is random we can say that it is difficult to have an instructor teach all the core courses of CSE.

****

**Query 6: Finding the Top 10 Courses with the Highest Enrollments**

****

**4)Performance Optimization**

When doing indexing and data partitioning we were not getting and desirable optimized result. The performance was to similar to the original. This is because the work from indexing is already beinged handled by denormalising the data when we make the MongoDB schema. Similarly data Partitioning helps when the data is very large and complex but the data that I have used is very simple and also is not large so these methods are not really helping us much

**Caching is implemented in the script using the `.cache()` function applied to the DataFrames that are repeatedly used across multiple queries.**

**How Caching is Implemented:**

- Caching is applied by calling the `.cache()` method on a DataFrame. This instructs Spark to store the results of the DataFrame in memory after the first time an action is performed on it.

- Once a DataFrame is cached, subsequent actions (e.g., filtering, selecting, aggregating) on the same DataFrame will retrieve the data from memory rather than recomputing or reloading it from MongoDB.

- Caching helps to avoid repeated computations and speeds up query execution, especially when the same data is reused multiple times across different queries.

**Where Caching is Implemented:**

**# Load collections and apply caching**

**students\_df = spark.read.format("mongodb") \**

**.option("database", database\_name) \**

**.option("collection", "students") \**

**.load().cache() # Cache the students dataframe**

**courses\_df = spark.read.format("mongodb") \**

**.option("database", database\_name) \**

**.option("collection", "courses") \**

**.load().cache() # Cache the courses dataframe**

**instructors\_df = spark.read.format("mongodb") \**

**.option("database", database\_name) \**

**.option("collection", "instructors") \**

**.load().cache() # Cache the instructors dataframe**

**departments\_df = spark.read.format("mongodb") \**

**.option("database", database\_name) \**

**.option("collection", "departments") \**

**.load().cache() # Cache the departments dataframe**

**Why Caching is Useful Here:**

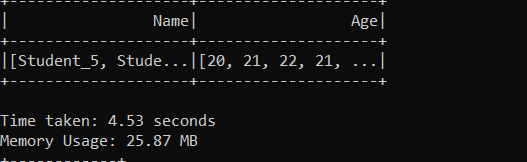
**1. Avoiding Repeated MongoDB Reads:** Since `courses\_df` and `students\_df` are used in multiple queries, caching prevents Spark from re-reading the entire collection from MongoDB for each query. Instead, the data is read once, cached, and reused for subsequent queries.

**2. Improved Performance:** Caching reduces the time spent recomputing or reloading data, especially when the same DataFrame is used in multiple operations.

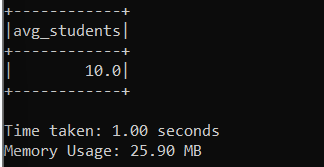
**3. Memory Efficiency:** Spark keeps cached DataFrames in memory, which provides much faster access compared to fetching data from MongoDB every time, especially when the same data is needed for multiple queries.

**Query Results and Performance Observations**

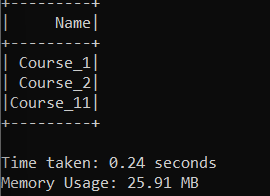
**Query 1: Fetching All Students Enrolled in a Specific Course**

****

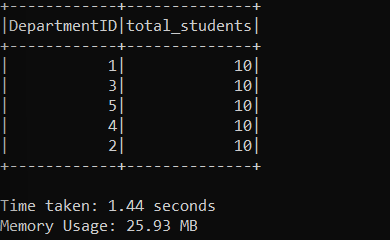
**Query 2: Calculating the Average Number of Students per Course for a Specific Instructor**

****

**Query 3: Listing All Courses Offered by a Specific Department**

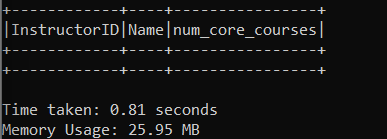
****

**Query 4: Finding the Total Number of Students per Department**

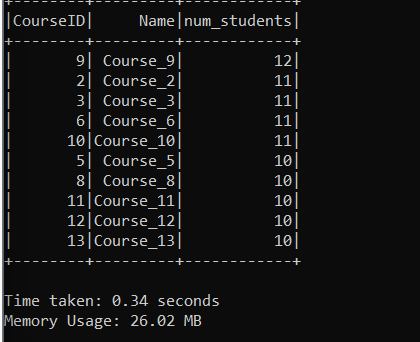
****

**Query 5: Listing Instructors Who Have Taught All Core Courses in BTech CSE**

No instructor teaches all the core courses of CSE department. As my data is random we can say that it is difficult to have an instructor teach all the core courses of CSE.

****

**Query 6: Finding the Top 10 Courses with the Highest Enrollments**

****